Consiglio Nazionale delle Ricerche

Exploiting Web Matrix Permutations to Speedup PageRank Computation

G. Del Corso, A. Gulli, F. Romani

IIT TR-04/2004

Technical report

Maggio 2004

Istituto di Informatica e Telematica

Exploiting Web Matrix Permutations to Speedup PageRank Computation

Gianna M. Del Corso[1] Antonio Gullí[1,2] Francesco Romani[1]

[1] Dipartimento di Informatica, University of Pisa, Italy
[2] IIT-CNR, Pisa

Abstract. Recently, the research community has devoted an increased attention to reduce the computational time needed by Web ranking algorithms. In particular, we saw many proposals to speed up the well-known PageRank algorithm used by Google. This interest is motivated by two dominant factors: (1) the Web Graph has huge dimensions and it is subject to dramatic updates in term of nodes and links - therefore PageRank assignment tends to became obsolete very soon; (2) many PageRank vectors need to be computed according to different personalization vectors chosen. In the present paper, we address this problem from a numerical point of view. First, we show how to treat dangling nodes in a way which naturally adapts to the random surfer model and preserves the sparsity of the Web Graph. This result allows to consider the PageRank computation as a sparse linear system in alternative to the commonly adopted eigenpairs interpretation. Second, we exploit the Web Matrix reducibility and compose opportunely some Web matrix permutation to speed up the PageRank computation. We tested our approaches on a Web Graphs crawled from the net. The largest one account about 24 millions nodes and more than 100 million links. Upon this Web Graph, the cost for computing the PageRank is reduced of 58% in terms of Mflops and of 89% in terms of time respect to the Power method commonly used.

Keywords: Link Analysis, Search Engines, Linear Systems, Eingepairs, Web Matrix Reducibility and Permutation.

1 Introduction

The Web is becoming a massive repository for delivering information. Recent studies [11] show that most of the users access the Web by means of a search engine. Nielsen/NetRatings, one of the leading Internet and digital media audience information and analysis services, reports that there were an estimated 151 million active Internet users in the US, for January 2004. Of these, 76 percent used a search engine at least once during the month and the average time spent searching was about 40 minutes. Therefore, to access something on Internet the easiest way is to insert some keywords on a favorite search engine and to jump to the selected Web site returned as answer. For these reasons, we witnessed to a growing interest and in large investment efforts to improve the quality Web ranking algorithms. Besides, the research community has devoted an increased attention to reduce the computation time needed by Web ranking algorithms.

Indeed, a particular attention was devoted to improve PageRank [31, 5], the well known ranking algorithm used by Google. The core of PageRank exploits an iterative weight assignment of ranks to the Web pages, until a fixed point is reached. This fixed point turns out to be the computation of the (dominant) eigenpairs of a matrix derived by the Web Graph itself. Brin and Page originally suggested to compute this pair using the well-known Power method [14] and they also gave a nice interpretation of PageRank in term of Markov Chains. The most recent researches about PageRank are aimed to address at least two different needs. First, the desire to reduce the time spent to weight the nodes of a Web Graphs containing many billions of pages which requires several days of computation. Note that Fetterly et al. in [13] showed that 35% of the Web changed during the course of their study (eleven downloads over a period of 11 weeks). More recently Cho et al. [11] showed that in a week more than 25% of links are changed and 5% of "new content" is created. They conclude that *"This result indicates that search engines need to update linkbased ranking metrics (such as PageRank) very often... a week-old ranking may not reflect the current ranking of the pages very well"*. The second need, is to assign *many* PageRanks values to each Web page, as results of PageRank's personalization [16, 17, 22, 10] that was recently presented by Google as beta-service [20].

Previous approaches to accelerating the PageRank followed three different directions. The first approach try to fit the graph structure into main memory by using compression techniques for the Web Graph [4, 32]. The second approach [15, 8], efficiently computes in external memory the PageRank, by using a sequence of scan operations on the Web graph. The last direction exploits efficient numerical methods to reduce the computation time. These kind of numerical techniques are the most promising and we have seen many intriguing results in the last few years. Arasu et al. [1] observe that computing PageRank is equivalent to solve a linear system and proposed to exploit iterative methods such as the Gauss-Seidel algorithm. They also suggest to reduce the computation time by exploiting the "bow-tie" structure of the Web [6]. Bianchini et al [3] propose to use Jacobi method to compute PageRank. Kamvar et al. [23] suggest to accelerate computation by using extrapolation methods which periodically subtracts off estimates of non-principal eigenvectors from the current iterate of the power method. The reported speedup is about 50-300% and it is influenced by the particular parameter settings used for computing the PageRank. A related results is given by Haveliwala et al. in [18], where they show how to compute the second eigenvalue associated to the PageRank. Kamvar et al. [25] note that the Web Graph's matrix, permuted by sorting the lexicographically the urls, assumes an approximate block structure since most of the links are intra-domains [2]. They suggest to compute separately several "local" PageRank vectors, one for each block and to use the concatenation of these vectors as starting point for the computation of a global web rank. In [24] the authors suggest to avoid the re-computation of the (sub)-components of PageRank vector as soon as they arrive to a fixed point. This results in a speedup of about 17%. Lee et al [28] obtain the most significant result. They suggest to split the PageRank in two sub-problems: one is obtained by collapsing in a single (super) node all the Web pages with no out-links, and the other is obtained combining in a single node all the pages with outgoing hyperlinks. The solutions of these two sub-problems are then combined to produce the PageRank vector. There is another technique presented in literature to address fast PageRank computation. As the Web Graph changes continuously, the key ideas is to update the PageRank of just those nodes which are influenced by such a change. In general the updating problem is a well-known problem for Markov Chains [26]. Nevertheless, at the moment the only know result for updating PageRank is [9] which just works in case of hyperlinks modification and not in the case of Web pages modification. We

should also note that, although there are some nice results [30] about the PageRank's stability in presence of small perturbation of the Web Graph, the results due to Fetterly [13] and Cho [11] shows, on the other hand, that the Web Graph is subject to rapid and *large* changes in terms of both nodes and links.

This paper contributes to numerical optimization of PageRank in number of ways viewing the PageRank computation as a linear system and applying optimization strategies for reducing the computational cost. In section 4 we show how to treat dangling nodes in a way which naturally adapts to the random surfer model. Although the computation of PageRank has already been seen as a linear system [1], we show how to handle the density of Web matrix by transforming a dense problem in one which uses a matrix as sparse as the Web Graph itself. In section 5, we deeply investigated the structure of the Web Matrix by exploiting its reducibility. This leads to adopting some permutation operators, composed opportunely, which rearrange the matrix in convenient shapes which increase the data locality and reduce number of iterations needed by the solving methods. In particular, we tested several scalar methods such as Power, Jacobi, Gauss-Seidel and Reverse Gauss-Seidel as well as block triangular methods. We test our approaches on a Web Graph crawled from the net of about 24 millions nodes and more than 100 millions of links. Our best result, achieved by a block method, is a reduction of 58% in Mflops and of 89% in time with the respect of the Power method commonly used to compute the PageRank.

2 Basic Definition and Notation

In this section we give some notation and definitions that will be useful in the rest of the paper. Let M by an $n \times n$ matrix. A scalar λ and a non-zero vector $\mathbf{x}$, are an eigenvalue and a corresponding (right) eigenvector of M if they are such that $M\mathbf{x} = \lambda\mathbf{x}$. In the same way, if $\mathbf{x}^T M = \lambda\mathbf{x}$, $\mathbf{x}$ is called left eigenvector corresponding to the eigenvalue λ. Note that, a left eigenvector is a (right) eigenvector of the transpose matrix.

A matrix is row-stochastic if its rows are non negative and the sum of each row is one. In this case, it is easy to show that there exists a dominant eigenvalue equal to 1 and a corresponding eigenvector $\mathbf{x} = (c, c, \ldots, c)^T$, for any constant c. A very simple method for the computation of the dominant eigenpair is the Power method [14] which, for stochastic irreducible matrices, is convergent for any choice of the starting vector with non negative entries. A stochastic matrix M can be viewed as a transition matrix associated to a family of Markov chains, where each entry M_{ij} represents the probability of a transition from state i to state j. By the Ergodic Theorem for Markov chains [33] an irreducible stochastic matrix M has a unique steady state distribution, that is a vector π such that $\pi^T M = \pi^T$. This means that the stationary distribution of a Markov chain can be determined by computing the left eigenvector of the stochastic matrix M. Given a graph $G = (V, E)$ and its adjacency matrix A we denote by outdeg(i) the out-degree of vertex i that is the number of non-zeros in the i-th row of A. Similarly the in-degree of vertex j, indeg(j), is the number of non-zeros in the j-th column of A. A node with no out-links is called dangling node.

3 The Google's PageRank Model

In this section we review the original idea of Google's PageRank [5]. The Web is viewed as an oriented graph (the Web Graph) where each of the N pages is a node and each hyperlink is an arc.

The intuition behind this model is that a page $i \in V$ is "important" if it is pointed by other pages which are in turn "important". This definition suggests an iterative fixed-point computation to assigning a rank of importance to each page in the Web. Formally, in the Page's model [31], a random surfer sitting on the page i can jump with equal probability $p_{ij} = 1/\text{outdeg(i)}$ to each page j adjacent to i. The iterative equation for the computation of PageRank $\mathbf{z}$ becomes

$$z_i = \sum_{j \in I_i} p_{ji} z_j,$$

where I_i is the set of nodes in-linking to the node i. The component z_i is the "ideal" PageRank of page i and it is then given by the sum of PageRanks assigned to the nodes pointing to i, weighted by the transition

probability p_{ij}. The equilibrium distribution of each state represents the ratio between the number of times the random walks passes over the state and the total number of transition, if it continues for infinite time. In matrix notation, the above equation is equivalent to the solution of the following system of equations

$$\mathbf{z}^T = \mathbf{z}^T P, \tag{1}$$

where $P_{ij} = p_{ij}$. This means that the PageRank vector $\mathbf{z}$ is the left eigenvector of P corresponding to the eigenvalue 1. In the rest of the paper, we assume that $||\mathbf{z}||_1 = \sum_{i=1}^{N} z_i = 1$, since the computation is not interested in assigning an exact value to each z_i, but rather in the relative order between the nodes.

The "ideal" model has unfortunately two problems. The first problem is due to the presence of dangling nodes. They capture the surfer indefinitely. Formally, a dangling node corresponds to an all-zero row in P. As a consequence, P is not stochastic and the Ergodic Theorem cannot be applied.

A convenient solution to the problem of dangling nodes is to define a matrix $\bar{P} = P + D$, where D is the rank one matrix defined as $D = \mathbf{d}\mathbf{v}^T$, where $\mathbf{d} = (d_i)_{i=1,\cdots,N}$ is

$$d_i = \begin{cases} 0 \text{ if outdeg(i)} \neq 0 \\ 1 \text{ if outdeg(i)} = 0, \end{cases}$$

and $\mathbf{v}$ is a *personalization vector* which records a generic surfer's preference for each page in V [31, 16, 24, 22]. The matrix $\bar{P}$ imposes a random jump to every other page in V whenever a dangling node is reached. Note that the new matrix $\bar{P}$ is stochastic. In section 4, we refer this model as the "natural" solution and compare it with other approaches proposed in the literature.

The second problem, with the "ideal" model is that the surfer can "get trapped" by a cyclic path in the Web Graph. Brin and Page [5] suggested to enforce irreducibility by adding a new set of artificial transitions that with low probability jump to all nodes. Mathematically, this corresponds to defining a matrix $\widehat{P}$ as

$$\widehat{P} = \alpha\bar{P} + (1-\alpha)\ \mathbf{e}\mathbf{v}^T, \tag{2}$$

where $\mathbf{e}$ is the vector with all entries equal to 1, and α is a constant, $0 < \alpha < 1$. At each step, with probability α a random surfer follows the transitions described by $\bar{P}$, while with probability $(1-\alpha)$ she/he bothers to follows links and jumps to any other node in V accordingly to the personalization vector $\mathbf{v}$. We remark that the treatment for dangling nodes in the natural models is in agreement with the idea behind the random jump. That is, when we encounter a dangling node we jump, in a natural way, with probability one to any other node. The matrix $\widehat{P}$ is stochastic and irreducible and both these conditions imply that the PageRank vector $\mathbf{z}$ is the unique steady state distribution of the matrix $\widehat{P}$ such that

$$\mathbf{z}^T\widehat{P} = \mathbf{z}^T. \tag{3}$$

From (2) it turns out that the matrix $\widehat{P}$ is explicitly

$$\widehat{P} = \alpha(P + \mathbf{d}\mathbf{v}^T) + (1-\alpha)\ \mathbf{e}\mathbf{v}^T. \tag{4}$$

The most convenient numerical method to solve the eigenproblem (3) is the Power method [14]. Since $\widehat{P}$ is a rank one modification of αP, it is possible to implement a power method which, at each step, multiply only the sparse matrix P by a vector and upgrades the intermediate result with a constant vector.

The eigenproblem (3) can be rewritten as a linear system. By substituting (4) into (3) we get

$$\mathbf{z}^T(\alpha P + \alpha\mathbf{d}\mathbf{v}^T) + (1-\alpha)\mathbf{z}^T\mathbf{e}\mathbf{v}^T = \mathbf{z}^T,$$

which means that the problem is equivalent to the solution of the following linear system of equations

$$S\mathbf{z} = (1-\alpha)\mathbf{v}, \tag{5}$$

where $S = I - \alpha P^T - \alpha\mathbf{v}\mathbf{d}^T$, and we make use of the fact that $\mathbf{z}^T\mathbf{e} = \sum_{i=1}^{N} z_i = 1$. The transformation of the eigenproblem (3) into the system (5) open the route to a large variety of numerical methods not completely investigated in literature. In next section we present a lightweight solution to handle the problem of the density of S.

4 Treatment of the Dangling Nodes

In this section we compare the different models proposed in literature for treating the dangling nodes while pointing out the benefits of the natural model. We remark that this issue is relevant since the dangling nodes can be a huge number. According to [25], a 2001 sample of the Web containing 290 millions pages had only 70 millions of non-dangling nodes. This large amount of nodes without outlinks includes both pages which do not point to any other page, and also pages whose existence is inferred by hyperlinks but not yet reached by the crawler. Besides, a dangling node can represent pdf, ps, txt or other any other file format gathered by a crawler but with no hyperlinks inside. Then, we show how we can compute the PageRank vector as the solution of a sparse linear system.

The first model of treatment proposed by Brin et al. [31] adopts the drastic solution of removing completely the dangling nodes. Doing that, the size of the problem is sensibly reduced but a large amount of information present in the Web is ignored. This has an impact on both the dangling nodes - which are simply not ranked - and on the remaining nodes - which don't take into account the contribute induced by the random jump from the set of dangling nodes. Moreover, removing this set of nodes could potentially create new dangling nodes, which must be removed in turn (see fig 1). Therefore, the nodes with an assigned PageRank could be a small percentage of the Web Graph.

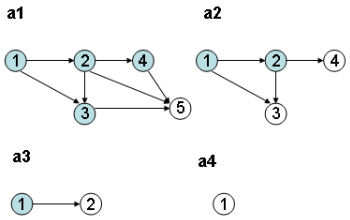


Fig. 1. Removing the dangling node in figure (a1) generates new dangling nodes, which are in turn removed (a2, a3, a4). At the end of the process no node receive a PageRank.

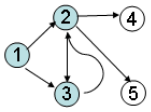
A different model was proposed by Arasu et al. [1]. They modify the Web Graph by imposing that every dangling node has a self loop. In term of matrices, $\bar{P} = P + F$ where

$$F_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and outdeg(i)} = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The matrix $\bar{P}$ is row stochastic and the computation of PageRank is solved using a random jump similar to the equation (4), where the matrix F replaces D. This model is different from the "natural" model as it is evident from the following example.

Example 1. Consider the below graph and the associated transition matrix. The PageRank obtained by using the natural model orders the node as follow (2, 3, 5, 4, 1), while the Arasu's model orders the node as follow (5, 4, 2, 3, 1). Note that in the latter case the node 5 receive a better rank than the node 2, which is counterintuitive. □

From the above observations we believe that it is important to take into account the dangling nodes and that the natural model is the one which better capture the behavior of a random surfer. The dense structure of the matrix S, caused by the presence of dangling nodes, poses serious problems to the solution of the linear system (5), this is why the PageRank vector has been mainly computed as solution of an eigenproblem where we can exploit the sparsity of the matrix P. In the following, we show how to handle dangling nodes in a direct and lightweight manner which make it possible to use iterative methods for linear systems. In particular, we



$$P = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

prove formally how the solution of the system (5) is equivalent to the solution of a system involving only the sparse matrix $R = I - \alpha P^T$. The following theorem holds.

Theorem 1. *The PageRank vector* $\mathbf{z}$ *solution of system* (5) *is obtained by solving the system* $R\mathbf{y} = \mathbf{v}$ *and taking* $\mathbf{z} = \gamma\mathbf{y}$ *in such a way* $||\mathbf{z}||_1 = 1$.

Proof. Since $S = R - \alpha\ \mathbf{v}\mathbf{d}^T$ equation (5) becomes

$$(R - \alpha\ \mathbf{v}\mathbf{d}^T)\mathbf{z} = (1 - \alpha)\ \mathbf{v}, \tag{7}$$

that is, a system of equations where the coefficient matrix is the sum of a matrix R and a rank-one matrix. Since R is non singular, we can use the well known Sherman-Morrison formula [19] for computing the inverse of the coefficient matrix. As a consequence, we have

$$(R - \alpha\mathbf{v}\mathbf{d}^T)^{-1} = R^{-1} + \frac{R^{-1}\mathbf{v}\mathbf{d}^T R^{-1}}{1/\alpha + \mathbf{d}^T R^{-1}\mathbf{v}}. \tag{8}$$

From (8), denoting by $\mathbf{y} = R^{-1}\mathbf{v}$, we have

$$\mathbf{z} = (1 - \alpha)\left(1 + \frac{\mathbf{d}^T\mathbf{y}}{1/\alpha + \mathbf{d}^T\mathbf{y}}\right)\mathbf{y},$$

that means that $\mathbf{z} = \gamma\mathbf{y}$, and the constant $\gamma = (1-\alpha)\left(1 + \frac{\mathbf{d}^T\mathbf{y}}{1/\alpha+\mathbf{d}^T\mathbf{y}}\right)$ can be computed normalizing $\mathbf{y}$ in such a way $||\mathbf{z}||_1 = 1$. □

Summarizing, we have shown that in order to compute the PageRank vector $\mathbf{z}$ we can just solve the linear system $R\mathbf{y} = \mathbf{v}$, and then normalize $\mathbf{y}$ to obtain the PageRank vector $\mathbf{z}$. This means that the introduction of the rank one matrix D in the PageRank model to account for dangling pages is algorithmically not necessary. Besides, we remark that the computation of the scaling factor γ is not necessary as well, since generally we are only interested in the relative order among the Web pages. We point out that the matrix used by Arasu and al. [1] is also sparse due to the way they deal with the dangling nodes. However, the PageRank obtained don't ranks the node in a natural way (see Example 1). Instead, our approach guarantees a more natural ranking and handles the density of S by transforming a dense problem in one which uses the sparse matrix R.

Our approach can be compared with the one proposed by Lee et al. [28]. They adopted the natural model, but they solve the eigenproblem (3) by constructing two independent Markov chains: one is obtained by collapsing in a single (super) node all the dangling nodes, and the other is obtained combining in a single node all the remaining pages. The solutions of these two Markov chains are then combined to produce the PageRank vector. The nice property of their approach is that the convergence of the dangling nodes chain is obtained in just three steps due to the low-rank structure of the transition matrix. Our strategy is based on the solution of a linear system and it ranks the dangling nodes in a direct way.

5 Exploiting the Web Matrix Permutations

In the previous section we have shown how to transform the linear system involving the dense matrix S into an equivalent linear system where the matrix R is as sparse as the Web Graph. Moreover, when solving a

linear system, particular attention must be devoted to the conditioning of the problem. It is easy to show [27], that the condition number in the 1-norm of S is $\mu_1(S) = \frac{1+\alpha}{1-\alpha}$, which means that the problem tends to become ill-conditioned as α goes to infinity. On the other hand, $\mu_1(R) \leqslant \mu_1(S)$ and experimentally we observed on many Web graphs that the system involving R is much better conditioned than the one involving S for α going to 1. One can always construct a graph and a matrix P for which the condition number of R and S is the same, but it can be proved that the above inequality is strict if the Web graph has at least a dangling node for each connected component.

To solve the linear system $R\mathbf{y} = \mathbf{v}$ the most convenient strategies are Jacobi, Gauss-Seidel or SOR methods, since they use a quantity of space comparable to the space used by the Power method. Aimed by the mentioned nice properties of R, we explored a range of different strategies for further discovering permutations to reduce the cost of linear system solution for the PageRank problem and to increase data locality on the Web Matrix. Note that the well know Jacobi method for sparse matrix linear system solution does not benefit from matrix permutation in terms of number of iterations to reach convergence. However, small differences in numerical data are due to the finite precision. In fact, for any matrix A and for any permutation matrix Π, the iteration matrix for Jacobi method of $B = \Pi A \Pi^T$ is $J_B = \Pi J_A \Pi^T$, which means $\rho(J_A) = \rho(J_B)$. A similar situation happens using the power method. Instead, for Gauss-Seidel or SOR iteration methods [34] opportune permutations can lead to iteration matrix with a reduced spectral radius. Our permutation strategies are obtained by combining different elementary and lightweight operations which have a limited impact on the computational cost. The elementary operator we defined are: ordering the nodes for increasing or decreasing out-degree, $\mathcal{O}(.)$ or $\mathcal{Q}(.)$ operator; ordering the nodes for increasing or decreasing in-degree, $\mathcal{X}(.)$ or $\mathcal{Y}(.)$ operator; permuting the nodes according to the BFS order of visit, $\mathcal{B}(.)$ operator which reduces the matrix into a lower block triangular form. Efficient algorithms for semi-external BFS on large graphs are given in [7, 29]. In [6] several properties of a Web Graph are studied using a BFS visit. Besides, we have also to transpose the matrix, which is done by the $\mathcal{T}(.)$ operator. The composition of the above operators give raise to the taxonomy shown in figure 2.

Full	Lower Block Triangular	Upper Block Triangular
$\mathcal{T}$	$\mathcal{TB}$	$\mathcal{BT}$
$\mathcal{OT}$	$\mathcal{OTB}$	$\mathcal{OBT}$
$\mathcal{QT}$	$\mathcal{QTB}$	$\mathcal{QBT}$
$\mathcal{XT}$	$\mathcal{XTB}$	$\mathcal{XBT}$
$\mathcal{YT}$	$\mathcal{YTB}$	$\mathcal{YBT}$

Fig. 2. Web Matrix Permutation Taxonomy.

In accordance with the taxonomy in figure 2, we denote, for instance, by $R_{\mathcal{XTB}} = I - \alpha\mathcal{B}(\mathcal{T}(\mathcal{X}(P)))$. Note that $\mathcal{T}$ operation is always required since the system involves the transpose of P. The first column in figure 2 gives raise to full matrices, while the second and third columns produce block triangular matrices due to the BFS's order of visit. In figure 5 we show a plot of the structure of the matrix R rearranged according to each item of the above taxonomy.

We have also investigated the effect of a symmetrization of the Web Matrix in order to exploit the effectiveness of a Cuthill-McKee like bandwidth reduction method [12]. Our experimental results do not show any significant benefits, compared to the space and time cost required to symmetrize the matrix and to construct the permutation. Figure 5 suggests that this approach can be promising for compressing the Web Graph and for increasing data locality. We plan to investigate this in future works.

We adopted ad hoc numerical methods for dealing with the different shapes of matrices in figure 5. In particular, we compared Power method, and Jacobi iterations with Gauss-Seidel, Reverse Gauss-Seidel. SOR and Adaptive SOR are currently investigated. Note that $R_{\mathcal{OT}} = JR_{\mathcal{QT}}J^T$ and $R_{\mathcal{XT}} = JR_{\mathcal{YT}}J^T$ where J is the anti-diagonal matrix, that is $J_{ij} = 1$ iff $i + j = n + 1$. This means that applying Gauss-Seidel to $R_{\mathcal{OT}}$ ($R_{\mathcal{XT}}$) is the same that applying Reverse Gauss-Seidel to $R_{\mathcal{QT}}$ ($R_{\mathcal{YT}}$). Besides, to further exploit the matrix reducibility we experimented with block methods. In particular, for the matrix $R_{\mathcal{OT}}$ in which the dangling

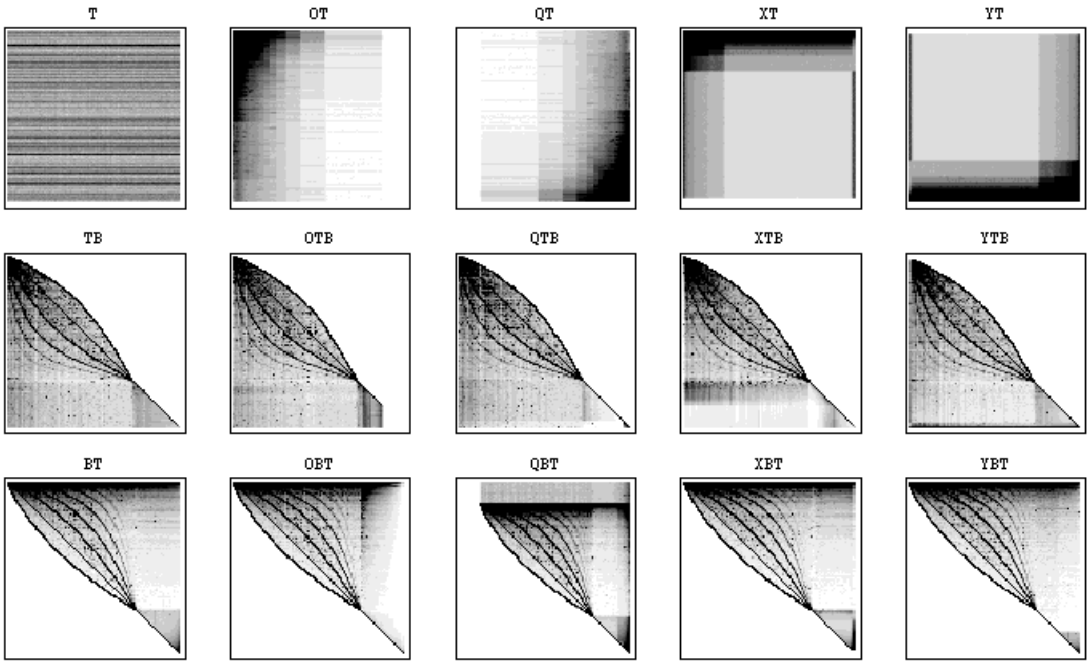


Fig. 3. Different shapes obtained by rearranging P in accordance to the taxonomy. First row represents full matrices; second and third lower and upper block triangular matrices respectively. Web Graph is made of 24 millions of nodes and 100 millions of links.

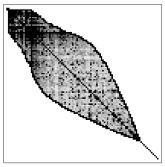


Fig. 4. A Web Matrix's shape after rearranging according to a permutation obtained applying an RCM-like method to $P + P^T$. The reduced bandwidth could be exploited as a Web graph's compression strategy.

nodes are listed in the last rows, a simple block forward substitution can be employed. The matrix $R_{\mathcal{OT}}$ has the nice property that the lower diagonal block coincides with the identity matrix. This means that the portion of the vector $\mathbf{y}$ relative to the dangling nodes doesn't require any matrix inversion. We denote this strategy as `DN` and `DNR` method depending on the use of Gauss-Seidel or Reverse Gauss-Seidel to solve the diagonal block. Another block strategy we explored, uses the fact that R is reducible as witnessed by the block triangular structure of the matrices transformed by the $\mathcal{B}(.)$ operator. This means that we can completely exploit the triangular block structure of the matrices. Each block is solved in the order they appear, either with Gauss-Seidel or Reverse Gauss-Seidel methods. The partial solutions are then combined using forward block triangular and backward block triangular solvers. For instance, on a lower block triangular system

$$\begin{bmatrix} R_{11} & & & \\ R_{21} & R_{11} & & \\ \vdots & & \ddots & \\ R_{m1} & \cdots & & R_{mm} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_m \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_m \end{bmatrix},$$

we have

$$\begin{cases} \mathbf{y}_1 = R_{11}^{-1}\mathbf{v}_1, \\ \mathbf{y}_i = R_{ii}^{-1}\left(\mathbf{v}_i - \sum_{j=1}^{i-1} R_{ij}\mathbf{y}_j\right) & \text{for } i = 2, \ldots, m. \end{cases}$$

Summing up, we have the taxonomy of solution strategies reported in figure 5.

Scalar methods	shapes
Jac	all
PM	all
GS	all
RGS	all

Block methods	shapes
DN	$R_{\mathcal{OT}}$
DNR	$R_{\mathcal{OT}}$
LB	$R_{*\mathcal{TB}}$
LBR	$R_{*\mathcal{TB}}$
UB	$R_{*\mathcal{BT}}$
UBR	$R_{*\mathcal{BT}}$

Fig. 5. Numerical Methods Taxonomy. Jac denotes the scalar Jacobi, PM is the Power method, GS and RGS are the Gauss-Seidel and Reverse Gauss-Seidel respectively. All of them can be applied to each transformation of the matrix according to the taxonomy 2. Among block-methods we have DN and DNR which partitions the node in dangling and non dangling solving the non dangling part with GS or RGS. Then, we have LB and LBR which can be applied to all lower block triangular matrices and uses GS or RGS to solve each diagonal block. Similarly, UB and UBR refers to the upper block triangular matrices.

6 Experimental Results

We validated the approaches discussed in previous section using three Web Graphs of different sizes. We report only the results on the largest one, which is a crawling of 24 millions of Web pages with about 100 millions of hyperlinks and containing approximately 3 millions of dangling nodes. This data set was donated to us by the Nutch project [21]. We run our experiments on a commodity PC with a Pentium IV 3Ghz, 2.0GB of memory and 512Mb of L1 cache.

A stopping criterion of 10^{-7} is imposed on the absolute difference between two successive iterations. In figure 6 we report the running time in seconds and the Mflops for each combination of solving (see table 5) and reordering (see table 2) methods. Some cells are empty since some methods are suitable only on particular shapes. Moreover, in figure 6 the results in the last two columns are relative to LB and LBR methods for lower block triangular matrices and UB or UBR for upper block triangular matrices. Note that for Jac and PM the Mflops are not influenced by the particular shape of the matrix, while the time expressed in seconds can change as results of different data locality.

Some comments on the numerical results are due. Jacobi method is essentially equivalent to Power Method commonly used for PageRank computation. The only difference is that PM has been applied to $\widehat{P}$ which incorporate the rank one modification accounting for dangling nodes even if they are handled using the standard optimizations suggested by [31, 15]. Instead, Jac method is applied to the sparse matrix R as resulting from Theorem 1. This motivate a reduction of Mflops of about 3%. GS and RGS have a better trend, but this is not surprising since the matrix R is a M-matrix [34] and Gauss-Seidel and Jacobi methods corresponds to regular partitioning. Hence it is a well-known result that $\rho(G_R) \leqslant \rho(J_R) < 1$, where G_R and J_R are the iteration matrices. GS and RGS account a reduced number of Mflops of about 40% and a time reduction of about 45%. These improvements are striking when combination of permutation operators are applied. The best performance is obtained using the $\mathcal{YTB}$ combination of operators on RGS method. This yields a Mflops reduction of 51% with respect to PM and a further reduction of 18% with respect to the GS both applied to the full matrix. In terms of time, the reduction is of 82%. Instead, the worst results in terms of Mflops is obtained using the $\mathcal{QTB}$ reordering and this corresponds to a increase of 16% on the GS applied to the full matrix. Nevertheless, the time expressed in seconds is considerably reduced to 61% and the data locality improved

in a significant way. Note that, RGS works better on lower block triangular matrices and GS works better on upper matrices. This is counterintuitive, but not surprising since the speed of converge depends on the spectral radius and not on the closeness of the matrices involved.

Name	Shape	Jac	PM	GS	RGS	DN	DNR	LB/UB	LBR/UBR
$\mathcal{T}$	Full	3080, 32105	3454, 33093	1903, 19957	2141, 20391	−, −	−, −	−, −	−, −
$\mathcal{OT}$	Full	2603, 32105	2934, 33093	1660, 20825	1784, 19957	1570, 19680	1515, 18860	−, −	−, −
$\mathcal{XT}$	Full	2940, 32105	3315, 33309	1920, 21259	1944, 19957	−, −	−, −	−, −	−, −
$\mathcal{TB}$	Lower Block	1094, 32105	1386, 32876	731, 21910	717, 18439	−, −	−, −	485, 16953	438, 15053
$\mathcal{OTB}$	Lower Block	1081, 32105	1383, 33093	705, 21476	705, 18439	−, −	−, −	480, 17486	446, 15968
$\mathcal{QTB}$	Lower Block	1049, 32105	1353, 32876	743, 23645	618, 16920	−, −	−, −	520, 18856	**385, 13789**
$\mathcal{XTB}$	Lower Block	1054, 32105	1361, 33309	682, 21259	715, 19090	−, −	−, −	484, 17196	472, 16414
$\mathcal{YTB}$	Lower Block	1100, 32105	1392, 32876	751, 22343	625, 16270	−, −	−, −	501, 17972	390, 13905
$\mathcal{BT}$	Upper Block	1102, 32105	1394, 33093	628, 18439	879, 22560	−, −	−, −	464, 15545	570, 19003
$\mathcal{OBT}$	Upper Block	1042, 32105	1341, 33309	605, 18873	806, 21693	−, −	−, −	470, 15937	543, 18312
$\mathcal{QBT}$	Upper Block	1211, 32105	1511, 33093	702, 18873	922, 21693	−, −	−, −	427, 15128	493, 17387
$\mathcal{XBT}$	Upper Block	1114, 32105	1408, 33093	667, 19306	860, 21693	−, −	−, −	474, 16075	541, 18265
$\mathcal{YBT}$	Upper Block	1059, 32105	1351, 33093	600, 18439	806, 21693	−, −	−, −	461, 15564	541, 18310

Fig. 6. Experimental Results: columns enumerate the numerical method exploited and rows enumerate the operators applied to the matrix R. Each cell represents the time in seconds and the number of megaflops taken by numerical method applied to the reordered matrix. Note that results in the last two columns account for the cost of the LB and LBR methods applied to Lower block Triangular matrices or for the cost of UB and UBR methods applied to Upper block triangular matrices. In bold we highlight our best result. We had a cost of only 42% in terms of Mflops with respect to the power method, commonly used to compute the PageRank.

Even better results are obtained by block methods. DN, which explores just the matrix reducibility due to dangling nodes, achieves a reduction of 41% in Mflops with respect to PM. The best among the results is obtained for the $\mathcal{QTB}$ reordering when the LBR solving method is applied, as depicted in figure 7. In this case, we have the very good reduction of 58% in Mflops and of 89% in terms of seconds required compared to PM on the full matrix commonly used to compute the PageRank. This means that our solving algorithm requires almost only a tenth of the time and much less than half of the Mflops of the Power Method.

The results given in table 6 doesn't take into account the effort spent in reordering the matrix. However, the most costly operator, namely the $\mathcal{B}$, can be efficiently implemented in semi external-memory as reported in [7, 29]. Our times for doing a BFS are comparable to those reported in [6] where less of 4 minutes are taken on a Web Graph with 100Millions nodes. The effort spent in applying permutation operators is largely repaid from the speedup achieved. Moreover, in case of personalized PageRank the permutations can be applied only once and reused for all personalized vectors $\mathbf{v}$ used.

7 Conclusion

The sheer and ever-growing size of Web Graph implies that the value and importance of fast methods for Web ranking is only going to rise in the future. In the light of the experimental results, our approach for speeding up PageRank computation appears much promising. We formalized a strategy for treating dangling nodes. It ranks the whole Web using a matrix which better capture the behavior of a random surfer with the respect to previous proposals [1]. We showed how to handle the density of this matrix by transforming a dense problem in one which uses a matrix as sparse as the Web Graph itself. This result allows to efficiently consider the PageRank computation as a sparse linear system, in alternative to the commonly adopted eigenpairs interpretation. Then, we discussed how to exploit the Web Matrix reducibility by composing opportunely

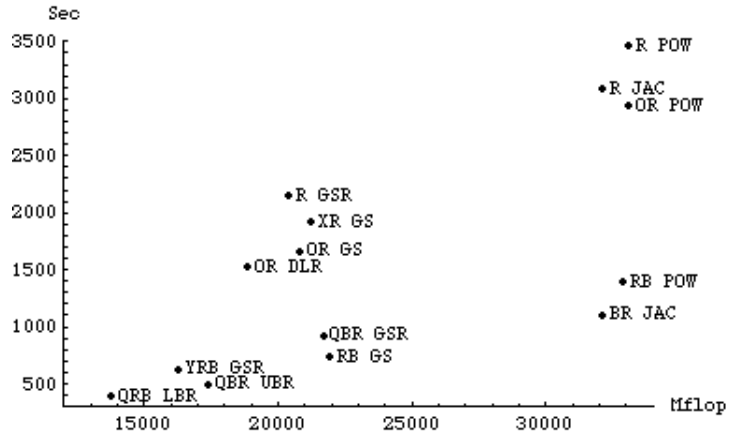


Fig. 7. Comparison

some Web matrix permutation operators to speed up the PageRank computation. We showed that permuting the Web matrix, according to a combination of in-degree or out-degree sorting order and BFS order, can effectively increase data locality and reduce the computation time when used in conjunction with numerical method such as lower block solvers. Our best result achieves a reduction of 58% in Mflops and of 89% in terms of seconds required compared to the Power Method commonly used to compute the PageRank. This means that our solving algorithm requires almost a tenth of the time and much less than half in terms of Mflops. The previous better improvement over the Power method is due to [28] where a reduction of 80% in time is achieved on a data set of roughly 400.000 nodes.

Acknowledgment

We thank Daniel Olmedilla of Learning Lab Lower Saxony, Doug Cutting and Ben Lutch of the Nutch Organization, Sriram Raghavan and Gary Wesley of Stanford University. They provided us some Web Graphs and a nice Web crawler. We thank Luigi Laura and Stefano Millozzi for their COSIN library. We also thank Paolo Ferragina for useful discussions and suggestions.

References

1. A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. PageRank computation and the structure of the Web: Experiments and algorithms. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.
2. K. Bharat, B.W. Chang, M. Henzinger, and M. Ruhl. Who links to whom: Mining linkage between Web sites. In *In Proceedings of the IEEE International Conference on Data Mining*, 2001.
3. M. Bianchini, M. Gori, and F. Scarselli. Inside PageRank. Technical report, University of Siena, 2003.
4. P. Boldi and S. Vigna. WebGraph framework i: Compression techniques. In *Proceedings of the Thirteenth International WWW Conference*, 2004.
5. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
6. A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener:. Graph structure in the Web. *Computer Networks*, 33:309–320, 2000.
7. Adam L. Buchsbaum, Michael Goldwasser, Suresh Venkatasubramanian, and Jeffery Westbrook. On external memory graph traversal. In *Symposium on Discrete Algorithms*, pages 859–860, 2000.
8. Y. Chen, Q. Gan, and T. Suel. I/o-efficient techniques for computing Pagerank. In *Proceedings of the Eleventh International WWW Conference*, 2002.

9. S. Chien, C. Dwork, S. Kumar, and D. Sivakumar. Towards exploiting link evolution, 2001.
10. P. Chirita, D. Olmedilla, and W. Nejdl. Pros: a personalized Ranking platform for Web search. Technical report, Learning Lab Lower Saxony, 2003.
11. J. Cho and S. Roy. Impact of Web search engines on page popularity. In *Proceedings of the Thirteenth International WWW Conference*, 2004.
12. E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *In Proc. 24th Nat. Conf. ACM*, pages 157–172, 1969.
13. D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of Web pages. In *Proceedings of the Twelfth International WWW Conference*, 2003.
14. G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, 1996. Third Edition.
15. T. Haveliwala. Efficient computation of PageRank. Technical report, Stanford University, 1999.
16. T. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the Eleventh International WWW Conference*, 2002.
17. T. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing PageRank. Technical report, Stanford University, 2003.
18. T. H. Haveliwala and S. D. Kamvar. The second eigenvalue of the Google matrix. Technical report, Stanford University, 2003.
19. A. S. Householder. *The Theory of Matrices in Numerical Analysis*. Blaisdell, New York, 1964.
20. http://labs.google.com/personalized/.
21. http://www.nutch.org/.
22. G. Jeh and J. Widom. Scaling personalized Web search. In *In Proceedings of the Twelfth International WWW Conference*, 2002.
23. S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Extrapolation methods for accelerating PageRank computations. In *Proceedings of the Twelfth International WWW Conference*, 2003.
24. S. D. Kamvar, T. H. Haveliwala, and G. H. Golub. Adaptive methods for the computation of PageRank. In *Proceedings of the International Conference on the Numerical Solution of Markov Chains*, 2003.
25. S. D. Kamvar, T. H. Haveliwala, C. Manning, and G. H. Golub. Exploiting the block structure of the Web for computing PageRank. Technical report, Stanford University, 2003.
26. A. N. Langville and C. D. Meyer. Updating the stationary vector of an irreducible Markov chain.
27. A. N. Langville and C. D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 2004. to appear.
28. C. P. Lee, G. H. Golub, and S. A. Zenios. A fast two-stage algorithm for computing PageRank. Technical report, Stanford University, 2003.
29. K. Mehlhorn and U. Meyer. External-memory breadthfirst search with sublinear I/O. In *European Symposium on Algorithms*, pages 723–735, 2002.
30. A. Y. Ng, A. X. Zheng, and M. I. Jordan. Stable algorithms for link analysis. In *Proc. 24th Annual Intl. ACM SIGIR Conference*, 2001.
31. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.
32. K. Randall, R. Stata, R. Wickremesinghe, and J. Wiener. The link database: Fast access to graphs of the Web. Technical report, Compaq Systems Research Center, 2001.
33. W. S. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1995.
34. R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, 1962.